(21)

SELF CONFIGURING STAGE DESCRIPTION

Revised March 1989

Michael L. Gordon

FILE: SELFCON SCRIPT
"

PRODIGY CONFIDENTIAL

TABLE OF CONTENTS

SUMMARY OF CHANGES

Note: Revisions made to this documents since its previous release are noted with a bar in the left margin.

November 1987	Original Release
December 1988	Support of sweeper function
March 1989	Support of large stages

SELF CONFIGURING STAGE

1.0 INTRODUCTION

This document describes the means of maintaining a dynamic collection of objects across sessions.

The motivation for this approach is enhanced response time, gained by avoiding repetitive object retrievals across sessions.

2.0 BENEFITS

- Following acquisition of an object, it will be retrieved again only if the version changes or it has been deleted from the stage for lack of use (and subsequently used again).
- The stage storage resource is better utilized in that actual usage will determine the individual stage's contents, rather than attempting to determine a least common denominator for all members.
- The interrogation of object version currency is driven by the individual reception system, which knows which objects need to be checked. As such, a universal update list with potentially a large percentage of irrelevant updates for the particular stage will not be transmitted.
- The version check occurs on demand, when the object is accessed for the first time during a session, ensuring currency.

3.0 OVERVIEW

3.1 GENERAL

Objects will be classified as to their reception system storage candidacy.

The reception system stage will be comprised of a small set of permanent objects, such as logon, that are needed before connection to the delivery server. All other objects that are stage candidates will be subject to an LRU algorithm determining which objects to forfeit from the stage as others are added. As

time goes on, the stage will be tailored to the particular behavior of the members using that system.

The dynamically staged objects need to be checked for currency before use. A new request to TOCS will support object version verification. In addition to the id of the object, this request will include the version of the object instance residing in the stage. TOCS will respond either with confirmation of currency or with the contents of the new version of the object.

An object that is considered to be particularly stable and not subject to change can be associated with the attribute of no version check. For such objects, no version checking will be performed. However, in order to stimulate version checking on such objects if the need arises, a control object which is version checked by the reception system following server connection will contain an updated NoVersionCheck Class version value. If this control object fails version checking and the Class version value differs from the previously stored one in the reception system, then a one shot version check attribute is associated with all existing objects in the stage that have the no version check attribute. As the individual objects are version checked, this one shot check attribute is removed and the object reverts to its previous behavior or newly defined one if a new instance of the object is delivered.

3.2 SWEEPER FUNCTION

Required objects which are not version checked, either by lack of reference or by the attribute of no version check without control object one shot version check stimulation, can accumulate in the stage inappropriately. For example, if a retired required object is not version checked, the deletion process would not take place and the object would linger in the stage. A solution is to sweep the stage over time, version checking all required objects. The method is to version check the least recently used required object during the session, promoting the object to the top of the usage list if still a stage candidate. One such object will be checked per session. Over time, all required objects will be version checked, thereby eliminating dead objects.

In order to work effectively, the version check attribute of the object needs to be ignored, so that any required object can be version checked. However, there may be times, such as during new diskette early deployment, when unconditional version checking may result in premature version checking. As such, a method will exist to disable the sweeper. A Sweeper Control Segment will be defined in the control object. A field will act as the on/off switch of the Sweeper.

3.2 LARGE STAGE

Large stages may be beneficial in environments that can support the disk and memory resources necessary for their existence. Potentially, more objects can be locally available for use. There is concern that if the universe of stage candidate objects is increased to exploit the increased storage capacity of the

large stages, there may be negative consequences to performance of the regular sized stage, such as thrashing.

When a stage is created, it will be assigned the classification of either a regular or large sized stage.

Two storage candidacy values will be created for objects which are candidates for a large stage, large stage candidacy with and without version checking. An object with either candidacy will be a stage candidate for a large stage and will be treated as a cacheable object if a regular sized stage is present.

4.0 LIMITATIONS

The set of stageable objects will be limited to those which are updated infrequently.

5.0 SYSTEMS COMPONENTS

5.1 Producer System

The version value field of the object header will be updated appropriately to indicate a new instance of an object.

The storage candidacy field of the object header will need to indicate the storage class of the object.

Both of these fields exist within the version identification field of the object header, described in the Systems Specification section.

5.2 Series/1

Support for new length of object request message.

5.3 TCS

Support for new length of object request message.

Support for old length message: pad with zeros to the new length in order to phase the implementation.

5.4 TOCS

Support for the object version verification feature. It will respond with either currency confirmation, or with the current object.

5.5 Reception System

Support for the dynamic stage management. Both LRU and optimized backing file update support will be added.

Support for the version id argument in the object request and currency confirmation response.

6.0 SYSTEMS SPECIFICATIONS

6.1 Object Version Identification

The version identifier will be a 16 bit unsigned integer derived from the following bytes of the object header:

- low byte of integer: byte 18 of header
- high byte of integer: byte 16 of header

The identifier is comprised of two fields:

BITS:	*BYTE BITS IN BYTE:*	*FIELD:*
[0..12]	*low [0..7]*	*version value*
	high [0..4]	
[13..15]	*high [5..7]*	*storage candidacy*

The version value field will be used to represent the particular version of an object and is the value that is varied as the instance of an object changes. The range of the field is zero through 8191.

The storage candidacy field defines the way in which the object can be stored on the reception device.

VALUE	MEANING:
0	*intrasession cache candidacy*
1	*no local residency after use*
2	*intersession stage candidacy, version checking enabled*
3	*intersession stage candidacy, no version checking*
4	*intersession stage required, version checking enabled*
5	*intersession stage required, no version checking*
6	*intersession large stage candidacy, no version checking*
7	*intersession large stage candidacy, version checking enabled*

6.2 Series/1

New object request length: current length plus 2.

6.2.1 Request Format: PC to Series/1

Fields HL through DID remain as currently defined.

BYTE SIZE	*DESCRIPTION*	*VALUE(S)*
2	*TL : Text length*	*x'000F'*
13	*Object ID*	*id of object*
2	*Version ID*	*id of version to check*

The version id is built as follows:

byte 1: low byte of id, byte 18 in object header
byte 2: high byte of id, byte 16 in object header

6.3 TCS

New object request length: current length plus 2.

Support for old length message: pad with zeros to the new length in order to phase the implementation.

6.4 TOCS

The object request will include the version id for comparison. If the current instance of the object available to TOCS (RAM cache, DASD, higher node server acquisition) has a version id equal to the one specified in the request, the response will be "Object Version Match." If the version ids do not match, the response will be the delivery of the new instance, identical to the existing successful response to the object request. If the object is not found, the response will be the existing "Object not Found" message.

If both bytes of the version id are set to binary zeros then no version check will occur, and the object will be delivered to the reception device.

6.4.1 Request Format: to TOCS

BYTE SIZE	*DESCRIPTION*	*VALUE(S)*
13	*Object ID*	*id of object*
1	*Request ID*	*id of Reception Request*
2	*Version ID*	*id of version to check* *0 implies no check*

The version id is built as follows:

byte 1: low byte of id, byte 18 in object header
byte 2: high byte of id, byte 16 in object header

6.4.2 Version Match Success Response

Same as single block "Get Object" successful response except the text length field is set to zero and no object data follows.

6.5 Reception System

6.5.1 Object Version Currency Check

When a staged object is initially fetched or accessed during a session, a server object request will be made specifying the version id of the instance locally resident. The server response will be: version match success, object data delivery, or object not found. If the version is current, the local instance will be used for the remainder of the session. If the object is delivered, the new contents will be used and the update installed in the stage. If the object is not found, the stage object entry will be deleted.

Requests for objects which are not locally resident will specify the binary zero values for the version id, resulting in unconditional object delivery.

6.5.2 Dynamic Stage Management

The stage storage facility will support the object operations of creation, update and deletion; and the Usage List update operation.

The disk IO required to modify the stage can result in performance penalties as well as media wear. As such, operations will be deferred, or batched, where possible. When the IO does take place, we will ensure that all pending updates are applied.

Additions will be made as objects are acquired from the server which are stage candidates. These objects will persist in the memory cache until such time as a memory cache forfeiture would take place, an upper bound count of changes has occurred, or the session will end.

Updates will be made when the version id of a staged object fails the currency check of the TOCS server, and the current instance of the object is received. As with additions, the object content updates will be batched.

Deletions will occur either as a demand for a stage resource stimulates forfeiture of an object or when an object not found response to a version currency check occurs.

The stage checkpointing will ensure that both the data and control information have been written to the backing storage media. Updating signatures will bound control structures so that incomplete media writes will be able to be detected.

7.0 TEST PLAN

7.1 Producer System

The correctness of the object database as made available to the delivery system needs to be confirmed.

Particular accuracy required:

- version value management, including rollover
- storage candidacy management
- stability of the above data, no contamination

The delivery database can and should be updated before the other components of the system are integrated.

7.2 Series/1, TCS and TOCS

These components can be integrated as a package for testing and installation.

Particular cases to be tested:

- continued support of current object requests
- support of version check match
  - objects in existence at TOCS
  - objects to be accessed from TPF server
- support of version check mismatch
- support of version field set to zero, unconditional object delivery

7.3 Reception System

This component can be tested independently and be the last to be installed, in effect activating the version check service.

The particular behaviors to be tested:

- usage list management of the stage
- forfeiture of best candidate from stage
- first time access version check request to TOCS server
- storage candidacy field appropriate behavior

7.3.1 PC or PS/2 based TOCS simulator

Support for version check requests and programmed responses.

8.0 RECEPTION CONTROL OBJECT DEFINITION

The Reception Control Object (RCO) will contain environmental data that is appropriate to distribute within object packaging.

This object, which is a required stage object, will be version checked when the first object request occurs following server connection. It can be acted upon by any of the components of the reception system, including programs packaged in objects.

Segments can be defined and added to this object as appropriate.

8.1 Object type

This is a data object.

8.2 Object name

| ITRC0001.D01, internally: "ITRC0001D <hex 01><hex 0C>"

8.3 Segments

The segment type is program data. The type field within the segment (following ST and SL) is either set to TBOL (1) or table data (2).

8.3.1 No Version Check Class version value

Segment type is program data, type field is table data, and sub-type field is No Version Check Class descriptor.

This segment will contain the current version value of the class of stage candidate objects for which no version checking is performed. A change of this version value is the stimulus to set the one shot version check attribute for currently staged objects belonging to this class.

8.3.1.1 Segment description

BYTE SIZE	*DESCRIPTION*	*VALUE(S)*
1	*ST: segment type*	*x'61' (program data)*
2	*SL: segment length*	*first byte: x'7'* *second byte: x'0'*
1	*Type of data*	*x'2' (table data)*
1	*Sub-type*	*x'1' (No Version Class)*
2	*Class Version Value*	*first byte: low 8 bits of value* *second byte: high 8 bits of value*

8.3.2 Sweeper Control

Segment type is program data, type field is table data, and sub-type field is Sweeper Control.

8.3.2.1 Segment Description

BYTE SIZE	*DESCRIPTION*	*VALUE(S)*
1	*ST: segment type*	*x'61' (program data)*
2	*SL: segment length*	*first byte: x'6'* *second byte: x'0'*

1	*Type of data*	*x'2' (table data)*
1	*Sub-type*	*x'2' (Sweeper Control)*
1	*Sweeper Switch*	*x'0' : disable sweeping operation* *x'1' : enable sweeping operation*